CS 598CSC: Approximation Algorithms	**Lecture date: February 4, 2009**
Instructor: Chandra Chekuri	**Scribe: Rachit Agarwal**

Fixed minor errors in Spring 2018.

In the last lecture, we studied the KNAPSACK problem. The KNAPSACK problem is an NP-hard problem but does admit a *pseudo-polynomial time* algorithm and can be solved efficiently if the input size is small. We used this pseudo-polynomial time algorithm to obtain an FPTAS for KNAPSACK. In this lecture, we study another class of problems, known as *strongly NP-hard* problems.

Definition 1 (Strongly NP-hard Problems) *An NPO problem π is said to be strongly NP-hard if it is NP-hard even if the inputs are polynomially bounded in combinatorial size of the problem* [1]*.*

Many NP-hard problems are in fact strongly NP-hard. If a problem Π is strongly NP-hard, then Π does not admit a pseudo-polynomial time algorithm. For more discussion on this, refer to [1, Section 8.3] We study two such problems in this lecture, MULTIPROCESSOR SCHEDULING and BIN PACKING.

1 Load Balancing / MultiProcessor Scheduling

A central problem in scheduling theory is to design a schedule such that the finishing time of the last jobs (also called *makespan*) is minimized. This problem is often referred to as the LOAD BALANCING, the MINIMUM MAKESPAN SCHEDULING or MULTIPROCESSOR SCHEDULING problem.

1.1 Problem Description

In the MULTIPROCESSOR SCHEDULING problem, we are given m identical machines $M_1, \ldots, M_m$ and n jobs $J_1, J_2, \ldots, J_n$. Job J_i has a processing time $p_i \geq 0$ and the goal is to assign jobs to the machines so as to minimize the maximum load[2].

1.2 Greedy Algorithm

Consider the following greedy algorithm for the MULTIPROCESSOR SCHEDULING problem which we will call GREEDY MULTIPROCESSOR SCHEDULING.

```
GREEDY MULTIPROCESSOR SCHEDULING:
Order (list) the jobs arbitrarily
For $i = 1$ to $n$ do
    Assign Job $J_i$ to the machine with least current load
    Update load of the machine that receives job $J_i$
```

[1]An alternative definition: A problem π is strongly NP-hard if every problem in NP can be polynomially reduced to π in such a way that numbers in the reduced instance are always written in unary

[2]The load of a machine is defined as the sum of the processing times of jobs that are assigned to that machine.

This algorithm is also called a *list scheduling* algorithm since we can view it as assigning a list of numbers (job sizes) to one of the m bins in a greedy fashion. We prove that the GREEDY MULTIPROCESSOR SCHEDULING algorithm gives a 2-approximation.

Theorem 1 GREEDY MULTIPROCESSOR SCHEDULING *algorithm gives a* $(2 - \frac{1}{m})$*-approximation for any list.*

To prove the theorem, we will make use of two lower bounds on the length of the optimal schedule:

Observation: $OPT \geq \dfrac{\sum_i p_i}{m}$

since, by the pigeonhole principle, some machine must have at least $\sum_i p_i/m$ amount of processing time assigned to it [3].

Observation: $OPT \geq \max_i p_i$

Proof of Theorem 1: For some fixed list, let L denote the makespan of the GREEDY MULTIPROCESSOR SCHEDULING algorithm. Let L_i denote the load of machine M_i and let M_{i^*} be the most heavily loaded machine in the schedule by GREEDY MULTIPROCESSOR SCHEDULING algorithm.

Let J_k be the last job assigned to M_{i^*}. Since GREEDY MULTIPROCESSOR SCHEDULING algorithm assigns a job to the machine that is least loaded, all machines must be loaded at least $L - p_k$ at the time of assigning J_k. Hence, we have:

$$\left(\sum_{i=1}^{n} p_i\right) - p_k \geq m\left(L - p_k\right) \tag{1}$$

which implies:

$$\begin{aligned}
L - p_k &\leq \frac{\left(\sum_{i=1}^{n} p_i\right) - p_k}{m} \\
\text{hence} & \\
L &\leq \frac{\left(\sum_{i=1}^{n} p_i\right)}{m} + p_k\left(1 - \frac{1}{m}\right) \\
&\leq OPT + OPT\left(1 - \frac{1}{m}\right) \\
&= OPT\left(2 - \frac{1}{m}\right)
\end{aligned}$$

where the third step follows from the two lower bounds on OPT. □

The above analysis is tight, *i.e.*, there exist instances where the greedy algorithm produces a schedule which has a makespan $(2 - 1/m)$ times the optimal. Consider the following instance: $m(m-1)$ jobs with unit processing time and a single job with processing time m. Suppose the

[3] One could also argue as follows: if this were not the case, each of the machines will be running for time less than $\sum_i p_i/m$. Given that the number of machines is m, the total load processed by the machines over optimal schedule will be strictly less than $\sum_i p_i$, which gives a contradiction.

greedy algorithm schedules all the short jobs before the long job, then the makespan of the schedule obtained is $(2m-1)$ while the optimal makespan is m. Hence the algorithm gives a schedule which has makespan $2-1/m$ times the optimal.

It may seem from the tight example above that an approximation ratio $\alpha < (2-1/m)$ could be achieved if the jobs are sorted before processing, which indeed is the case. The following algorithm, due to [3], sorts the jobs in decreasing order of processing time prior to running GREEDY MULTIPROCESSOR SCHEDULING algorithm.

```
MODIFIED GREEDY MULTIPROCESSOR SCHEDULING:
Sort the jobs in decreasing order of processing times
For $i = 1$ to $n$ do
    Assign Job $J_i$ to the machine with least current load
    Update load of the machine that receives job $J_i$
```

Theorem 2 MODIFIED GREEDY MULTIPROCESSOR SCHEDULING *algorithm gives a* $(4/3 - 1/3m)$*-approximation for the* MULTIPROCESSOR SCHEDULING *problem.*

Claim 3 *Suppose* $p_i \geq p_j$ *for all* $i > j$ *and* $n > m$. *Then,* $OPT \geq p_m + p_{m+1}$.

Proof Sketch. Since $n > m$ and the processing times are sorted in decreasing order, some two of the $(m+1)$ largest jobs must be scheduled on the same machine. Notice that the load of this machine is at least $p_m + p_{m+1}$. □

Exercise: Prove that MODIFIED GREEDY MULTIPROCESSOR SCHEDULING gives a $(3/2 - 1/2m)$-approximation.

Before going to the description of a PTAS for MULTIPROCESSOR SCHEDULING problem, we discuss the case when the processing times of the jobs are bounded from above.

Claim 4 *If* $p_i \leq \varepsilon \cdot OPT$, $\forall i$, *then* MODIFIED GREEDY MULTIPROCESSOR SCHEDULING *gives a* $(1+\varepsilon)$*-approximation.*

1.3 A PTAS for Multi-Processor Scheduling

We will now give a PTAS for the problem of scheduling jobs on identical machines. We would like to use the same set of ideas that were used for the KNAPSACK problem (see Lecture 4): that is, given an explicit time T we would like to round the job lengths and use dynamic programming to see if they will fit within time T. Then the unrounded job lengths should fit within time $T(1+\varepsilon)$.

1.3.1 Big Jobs, Small Jobs and Rounding Big Jobs

For the discussion that follows, we assume that all the processing times have been scaled so that $OPT = 1$ and hence, $p_{max} \leq 1$.

Given all the jobs, we partition the jobs into two sets: *Big jobs* and *Small jobs*. We call a job J_i "big" if $p_i \geq \varepsilon$. Let $\mathcal{B}$ and $\mathcal{S}$ denote the set of big jobs and small jobs respectively, *i.e.*, $\mathcal{B} = \{J_i : p_i \geq \varepsilon\}$ and $\mathcal{S} = \{J_i : p_i < \varepsilon\}$. The significance of such a partition is that once we pack the jobs in set $\mathcal{B}$, the jobs in set $\mathcal{S}$ can be greedily packed using list scheduling.

Claim 5 *If there is an assignment of jobs in $\mathcal{B}$ to the machines with load L, then greedily scheduling jobs of $\mathcal{S}$ on top gives a schedule of value no greater than* $\max\{L, (1+\varepsilon)OPT\}$.

Proof: Consider scheduling the jobs in $\mathcal{S}$ after all the jobs in $\mathcal{B}$ have been scheduled (with load L). If all of these jobs in $\mathcal{S}$ finish processing by time L, the total load is clearly no greater than L.

If the jobs in $\mathcal{S}$ can *not* be scheduled within time L, consider the last job to finish (after scheduling the small jobs). Suppose this job starts at time T'. All the machines must have been fully loaded up to T', which gives $OPT \geq T'$. Since, for all jobs in $\mathcal{S}$, we have $p_i \leq \varepsilon \cdot OPT$, this job finishes at $T' + \varepsilon \cdot OPT$. Hence, the schedule can be no more than $T' + \varepsilon \cdot OPT \leq (1+\varepsilon)OPT$, settling the claim. □

1.3.2 Scheduling Big Jobs

We concentrate on scheduling the jobs in $\mathcal{B}$. We round the sizes of all jobs in $\mathcal{B}$ using geometrically increasing interval sizes using the following procedure:

```
ROUNDING JOBS:
For each big job $i$ do
    If $p_i \in (\varepsilon(1+\varepsilon)^j, \varepsilon(1+\varepsilon)^{j+1}]$
        Set $p_i = \varepsilon(1+\varepsilon)^{j+1}$
```

Let $\mathcal{B}'$ be the set of new jobs.

Claim 6 *If jobs in $\mathcal{B}$ can be scheduled with load 1, then the rounded jobs in $\mathcal{B}'$ can be scheduled with load* $(1+\varepsilon)$.

Claim 7 *The number of distinct big job sizes after rounding is* $O(\log_{1+\varepsilon}\varepsilon)$

Proof: Notice that due to scaling, we have $p_i \leq 1$ for all jobs J_i. Hence, we have:

$$\begin{aligned} \varepsilon(1+\varepsilon)^k &\leq 1 \\ \Rightarrow k &\leq \log_{(1+\varepsilon)}\frac{1}{\varepsilon} \end{aligned}$$

□

Lemma 8 *If the number of distinct job sizes is k, then there is an exact algorithm that returns the schedule (if there is one) and runs in time $O(n^{2k})$.*

Proof: Use Dynamic Programming. (Exercise, see Appendix A for the solution) □

Corollary 9 *Big Jobs can be scheduled (if possible) with load $(1+\varepsilon)$ in time $n^{O(\log_{1+\varepsilon}\frac{1}{\varepsilon})}$.*

Once we have scheduled the jobs in $\mathcal{B}$, using Claim 5, we can pack small items using greedy link scheduling. The overall algorithm is then given as:

```
PTAS MULTIPROCESSOR SCHEDULING:
1. Guess OPT
2. Define $\mathcal{B}$ and $\mathcal{S}$
3. Round $\mathcal{B}$ to $\mathcal{B}'$
4. If jobs in $\mathcal{B}'$ can be scheduled in $(1+\varepsilon)OPT$
        Greedily pack $\mathcal{S}$ on top
   Else
        Modify the guess and Repeat.
```

In the following subsection, we comment on the guessing process.

1.3.3 Guessing

We define a $(1+\varepsilon)$-relaxed decision procedure:

Definition 2 *Given $\varepsilon > 0$ and a time T, a $(1+\varepsilon)$-relaxed decision procedure returns:*

- *Output a schedule (if there is one) with load $(1+\varepsilon) \cdot T$ in time $n^{O(\frac{\log(1/\varepsilon)}{\varepsilon})}$*
- *Output correctly that there is no schedule of load T.*

Define

$$L = \max\left\{\max_j p_j, \frac{1}{m}\sum_j p_j\right\}$$

L is a lower bound on OPT using the two lower bounds on OPT discussed earlier. Furthermore, an upper bound on OPT is given by the GREEDY MULTIPROCESSOR SCHEDULING algorithm, which is $2L$. Our algorithm will perform *binary search* on $[L, 2L]$ with the decision procedure above until interval length shrinks to size $\varepsilon \cdot L$.

1.4 Section Notes

MULTIPROCESSOR SCHEDULING is NP-hard as we can reduce 2-PARTITION to MULTIPROCESSOR SCHEDULING on two machines. Note that this reduction only proves that MULTIPROCESSOR SCHEDULING is weakly NP-hard. This infact is the case when there are a constant number of machines. Horowitz and Sahni [4] give an FPTAS for this variant of the problem. However, the MULTIPROCESSOR SCHEDULING problem is strongly NP-hard for m arbitrary (by a reduction from 3-PARTITION [5]). Thus, there can not exist an FPTAS for the MULTIPROCESSOR SCHEDULING problem in general, unless P = NP. However, Hochbaum and Shmoys [6] gave a PTAS (a $(1+\varepsilon)$-approximation algorithm which runs in polynomial time for any fixed ε) for the problem.

2 Bin Packing

2.1 Problem Description

In the BIN PACKING problem, we are given a set of n items $\{1, 2, \ldots, n\}$. Item i has size $s_i \in (0, 1]$. The goal is to find a minimum number of bins of capacity 1 into which all the items can be packed.

One could also formulate the problem as partitioning $\{1, 2, \ldots, n\}$ into k sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$ such that $\sum_{i \in \mathcal{B}_j} s_i \leq 1$ and k is minimum.

2.2 Greedy Approaches

Consider the following greedy algorithm for bin packing:

```
GREEDY BIN PACKING:
Order items in some way
For $i = 1$ to $n$
  If item $i$ can be packed in some *open* bin
      Pack it
  Else
      Open a new bin and pack $i$ in the new bin
```

In GREEDY BIN PACKING algorithm, a new bin is opened only if the item can not be packed in any of the already opened bins. However, there might be several opened bins in which the item i could be packed. Several rules could be formulated in such a scenario:

- *First Fit*: Pack item in the earliest opened bin
- *Last Fit*: Pack item in the last opened bin
- *Best Fit*: Pack item in the bin that would have least amount of space left after packing the item
- *Worst Fit*: Pack item in the bin that would have most amount of space left after packing the item

Irrespective of what strategy is chosen to pack an item in the opened bins, one could get the following result:

Theorem 10 *Any greedy rule yields a 2-approximation.*

Observation: $OPT \geq \sum_i s_i$

We call a bin *α-full* if items occupy space at most α.

Claim 11 *Greedy has at most 1 bin that is $\frac{1}{2}$-full.*

Proof: For the sake of contradiction, assume that there are two bins B_i and B_j that are $\frac{1}{2}$-full. WLOG, assume that GREEDY BIN PACKING algorithm opened bin B_i before B_j. Then, the first item that the algorithm packed into B_j must be of size at most $\frac{1}{2}$. However, this item could have been packed into B_i since B_i is $\frac{1}{2}$-full. This is a contradiction to the fact that GREEDY BIN PACKING algorithm opens a new bin if and only if the item can not be packed in any of the opened bins. □

Proof of Theorem 10: Let m be the number of bins opened by GREEDY BIN PACKING algorithm. From Claim 11, we have:

$$\sum_i s_i > \frac{m-1}{2}$$

Using the observation that $OPT \geq \sum_i s_i$, we get:

$$OPT > \frac{m-1}{2}$$

which gives us:

$$\begin{aligned} m &< 2 \cdot OPT + 1 \\ \Rightarrow m &\leq 2 \cdot OPT \end{aligned}$$

□

2.3 PTAS for Bin Packing

A natural question follows the discussion above: Can BIN PACKING have a PTAS? In this subsection, we settle this question in negative. In particular, we give a reduction from an NP-complete problem to the BIN PACKING problem and show that a PTAS for the BIN PACKING problem will give us an exact solution for the NP-complete problem in polynomial time. We consider the PARTITION problem:

In the PARTITION problem, we are given a set of items $\{1, 2, \ldots, n\}$. Item i has a size s_i. The goal is to partition the $\{1, 2, \ldots, n\}$ into two sets $\mathcal{A}$ and $\mathcal{B}$ such that $\sum_{i \in \mathcal{A}} s_i = \sum_{j \in \mathcal{B}} s_j$.

Claim 12 *If* BIN PACKING *has a* $(\frac{3}{2} - \varepsilon)$*-approximation for any* $\varepsilon > 0$*, the* PARTITION *problem can be solved exactly in polynomial time.*

Proof: Given an instance I of the PARTITION problem, we construct an instance of the BIN PACKING problem as follows: Scale the size of the items such that $\sum_i s_i = 2$. Consider the scaled sizes of the items as an instance I' of the BIN PACKING problem. If all items of I' can be packed in 2 bins, then we have an "yes" answer to I. Otherwise, the items of I' need 3 bins and the answer to I is "no".

OPT for I' is 2 or 3. Hence, if there is a $(\frac{3}{2} - \varepsilon)$-approximation algorithm for the BIN PACKING problem, we can determine the value of OPT which in turn implies that we can solve I. Thus, there can not exist a $(\frac{3}{2} - \varepsilon)$-approximation algorithm for the BIN PACKING problem, unless P = NP. □

Recall the scaling property from Lecture 2, where we discussed why some of the optimization problems do not admit additive approximations. We notice that the BIN PACKING problem does not have the scaling property. Hence it may be possible to find an additive approximation algorithms. We state some of the results in this context:

Theorem 13 (Johnson '74 [7]) *There exists a polynomial time algorithm such* $\mathcal{A}_J$ *such that:*

$$\mathcal{A}_J(I) \leq \frac{11}{9} OPT(I) + 4$$

for all instances I of the BIN PACKING *problem.*

Theorem 14 (Fernandez de la Vega, Lueker '81 [8]) *There exists a polynomial time algorithm such* $\mathcal{A}_{FL}$ *such that:*

$$\mathcal{A}_{FL}(I) \leq (1 + \varepsilon) OPT(I) + 1$$

for all instances I of the BIN PACKING *problem and any* $\varepsilon > 0$.

Theorem 15 (Karmarkar, Karp '82 [9]) *There exists a polynomial time algorithm such* $\mathcal{A}_{KK}$ *such that:*

$$\mathcal{A}_{KK}(I) \leq OPT(I) + O(\log^2(OPT(I)))$$

for all instances I of the BIN PACKING *problem.*

Exercise: Show that *First Fit* greedy rule yields a $\frac{3}{2}OPT + 1$-approximation.

Open Problem: Does there exist a polynomial time algorithm $\mathcal{A}$ such that $\mathcal{A}(I) \leq OPT(I) + 1$? Is there an algorithm $\mathcal{A}$ such that $\mathcal{A}(I) \leq OPT(I) + c$, for some constant c?

2.4 Asymptotic PTAS for Bin Packing

A recurring theme in last two lectures has been the rounding of jobs/tasks/items. To construct an asymptotic PTAS for BIN PACKING problem, we use the same set of ideas with simple modifications. In particular, we divide the set of items into *big* and *small* items and concentrate on packing the big items first. We show that such a technique results in an asymptotic PTAS for the BIN PACKING problem.

Consider the set of items, $s_1, s_2, \ldots, s_n$. We divide the items into two sets, $\mathcal{B} = \{i \, : \, s_i \geq \varepsilon\}$ and $\mathcal{S} = \{j \, : \, s_j < \varepsilon\}$. Similar to the MULTIPROCESSOR SCHEDULING problem, where we rounded up the processing times of the jobs, we round up the sizes of the items in the BIN PACKING problem. Again, we concentrate only on the items in $\mathcal{B}$. Let $n' = |\mathcal{B}|$ be the number of big items.

Observation: $OPT \geq \varepsilon \cdot n'$

Claim 16 *Suppose $n' < 4/\varepsilon^2$. An optimal solution for the* BIN PACKING *problem can be computed in $2^{O(1/\varepsilon^4)}$ time.*

Proof Sketch. If the number of big items is small, one can find the optimal solution using brute force search. □

Claim 17 *Suppose $n' > 4/\varepsilon^2$. Then $OPT \geq 4/\varepsilon$.*

The following gives a procedure to round up the items in $\mathcal{B}$:

ROUNDING ITEM SIZES:
Sort the items such that $s_1 \geq s_2 \geq \cdots \geq s_{n'}$
Group items in $k = 2/\varepsilon^2$ groups $\mathcal{B}_1, \ldots, \mathcal{B}_k$ such that each group has $\lfloor n'/k \rfloor$ items
Round the size of all the items in group $\mathcal{B}_i$ to the size of the smallest item in $\mathcal{B}_{i-1}$

Lemma 18 *Let $\varepsilon > 0$ be fixed, and let K be a fixed non-negative integer. Consider the restriction of the bin packing problem to instances in which each item is of size at least ε and the number of distinct item sizes is K. There is a polynomial time algorithm that optimally solves this restricted problem.*

Proof Sketch. Use Dynamic Programming. (Exercise, Hint: How can you modify the algorithm of Appendix A?) □

Claim 19 *The items in $\mathcal{B}$ can be packed in $OPT + |\mathcal{B}_1|$ bins in time $n^{O(1/\varepsilon^2)}$.*

Proof: Using ROUNDING ITEM SIZES, we have restricted all items but those in $\mathcal{B}_1$ to have one of the $k-1$ distinct sizes. Using lemma 18, these items can be packed efficiently in OPT. Furthermore, the items in $\mathcal{B}_1$ can always be packed in $|\mathcal{B}_1|$ bins. Hence, the total number of bins is $OPT + |\mathcal{B}_1|$.

The running time of the algorithm follows from the discussion in Appendix A. □

Lemma 20 *Let $\varepsilon > 0$ be fixed. Consider the restriction of the bin packing problem to instances in which each items is of size at least ε. There is a polynomial time algorithm that solves this restricted problem within a factor of $(1 + \varepsilon)$.*

Proof: Using Claim 19, we can pack $\mathcal{B}$ in $OPT + |\mathcal{B}_1|$ bins. Recall that $|\mathcal{B}_1| = \lfloor n'/k \rfloor \leq \varepsilon^2 \cdot n'/2 \leq \varepsilon \cdot OPT/8$ where, we have used Claim 17 to reach the final expression. □

Theorem 21 *For any ε, $0 < \varepsilon < 1/2$, there is an algorithm $\mathcal{A}_\varepsilon$ that runs in time polynomial in n and finds a packing using at most $(1 + 2\varepsilon)OPT + 1$ bins.*

Proof: Assume that the number of bins used to pack items in $\mathcal{B}$ is m and the total number of bins used after packing items in $\mathcal{S}$ is m'. Clearly

$$m' \leq \max\left\{m, \left\lceil \frac{(\sum_i s_i)}{(1-\varepsilon)} \right\rceil\right\}$$

since at most one bin must be $(1 - \varepsilon)$ full using an argument in GREEDY BIN PACKING. Furthermore,

$$\left\lceil \frac{(\sum_i s_i)}{(1-\varepsilon)} \right\rceil \leq \left(\sum_i s_i\right)(1 + 2\varepsilon) + 1$$

for $\varepsilon < 1/2$. This gives the required expression. □

The algorithm is summarized below:

```
ASYMPTOTIC PTAS BIN PACKING:
Split the items in $\mathcal{B}$ (big items) and $\mathcal{S}$ (small items)
Round the sizes of the items in $\mathcal{B}$ to obtain constant number of item sizes
Find optimal packing for items with rounded sizes
Use this packing for original items in $\mathcal{B}$
Pack items in $\mathcal{S}$ using GREEDY BIN PACKING.
```

2.5 Section Notes

An excellent survey on approximation algorithms for BIN PACKING problem is [10].

References

[1] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, New York, NY, 2001

[2] R. L. Graham, Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563-1581, 1966.

[3] R. L. Graham, Bounds for multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416-429, 1969.

[4] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *Journal of ACM*, 23(2): 317-327, 1976

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, NY, USA, 1979

[6] D. S. Hochbaum and D. B. Shmoys, A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach, *SIAM Journal on Comp.*, 17(3): 539-551, 1988

[7] D. S. Johnson, Approximation Algorithms for Combinatorial Problems, *Journal of Computer System Science*, 9:256-278, 1974.

[8] W. F. de la Vega and G. S. Lueker, Bin packing can be solved within 1 + ε in linear time, *Combinatorica*, 1:349355, 1981

[9] N. Karmarkar and R. M. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science*, 312-320, 1982

[10] E. G. Coffman Jr., M. R. Garey and D. S. Johnson, *Approximation algorithms for bin backing: a survey*, In D.S. Hochbaum, editor, Approximation Algorithms for NP-Hard Problems, pages 4693. PWS Publishing, Boston, MA, 1997

A Scheduling Jobs when Number of Distinct Job Sizes is Constant

We give a dynamic programming algorithm:

Let k_i be the number of jobs of size i and let $(m_1, m_2, \ldots, m_k)$ be an arbitrary set of jobs. Given a target T, let $M(m_1, m_2, \ldots, m_k)$ denote the number of machines needed to schedule the jobs in $(m_1, m_2, \ldots, m_k)$ in time at most T. Assume that b_i is the number of big jobs of size i. Consider the following algorithm:

```
SCHEDULING BIG JOBS:
Set $M(0, 0, \ldots, 0) = 0$
For $k_1 = 1$ to $b_1$ do
  For $k_2 = 1$ to $b_2$ do
      ⋮
        For $k_k = 1$ to $b_k$ do
          If $M(k_1, k_2, \ldots, k_k) > 1$
            Let $M_1 = \{(j_1, j_2, \ldots, j_k), : j_i \le k_i, M(j_1, j_2, \ldots, j_k) = 1\}$
            $M(k_1, k_2, \ldots, k_k) = 1 + \min_{(j_1, j_2, \ldots, j_k) \in M_1} M(k_1 - j_1, k_2 - j_2, \ldots, k_k - j_k)$
If $M(b_1, b_2, \ldots, b_k) \le m$
  Return Yes
Else
  Return No
```

The algorithm runs in time $\mathcal{O}(n^{2k})$ since the innermost statement of the nested loop is executed as most $\mathcal{O}(n^k)$ times, each execution taking $\mathcal{O}(n^k)$ time. Moreover, by keeping track of the decisions made in the innermost statement (which can be done in linear time), one could output the schedule.